AUCO Czech Economic Review 2 (2008) 268–300
Acta Universitatis Carolinae Oeconomica

Algorithmic Approaches to Game-theoretical Modeling and Simulation

Martin Hrubý*

Received 9 October 2008; Accepted 7 December 2008

Abstract This paper deals with a methodology of computer modeling and simulation of market competitive situations using game theory. The situations are thematically focused mostly to models of commodity markets but the applications of the methodology can be wider. This methodology covers the whole modeling work, including a primary specification of a problem, making an abstract model, making a simulation model, design of a state space of the problem and the simulator itself. As a whole, the methodology represents a complete framework for implementation of computer models of commodity markets suitable for their further analysis and prediction of their future evolution. The main contribution of the paper consists in the algorithmic implementation of computer processing of large strategic game.

Keywords Market models, non-cooperative game theory, modeling and simulation, artificial intelligence

JEL classification C51, C53, C63, C72

1. Introduction

Game theory is traditionally considered to be a discipline of pure mathematicians and mathematically oriented economists. The literature on game theory is quite large and includes various (analytical) models of elementary situations. These models describe every-day decision problems of people playing games, traders doing their business, politicians in elections and others. But the scale factor of quality and advisability of every theory is expressed by its real applications arising from real requirements. In this point, the literature is not so rich and practical case studies can be found mostly in commercial papers advertising a particular analysis tool.

Research connected to this paper was motivated by colleagues from EGÚ Brno company. They are experts in analyzing and predicting the electricity markets which they do using their own computer models. In the recent years, we created a package of interconnected computer models covering the electricity markets in the region of Central Europe (Hrubý and Toufar 2006). Our models include several aspects of the electricity market commodities, behavior of the producers, consumers and the overall legislation. Although the architecture of the mentioned models might be of interest of the readers, their detail description would exceed the size of a journal paper. For that reason, this particular paper focuses mostly to the mathematical and modeling

* Brno University of Technology, Faculty of Information Technology, Božetěchova 2, 612 66 Brno, Czech Republic. Phone +420 54114-1140, E-mail: hrubym@fit.vutbr.cz.

fundamentals of our models. It deals with methods of applying the game theoretical principles in computer science. From the computer science point of view, we work on artificial intelligence in the computer modeling. The main contribution of the paper is supposed to be in the presented methodology of designing the computer models and in the algorithms of their simulation. It might be an inspiration for those who like to implement the game theory in practical computer models. Anyway, as the world of electricity is very close to our thinking, we will recall the terminology of the electricity markets to demonstrate the described ideas in suitable moments.

Computer modeling and simulation together with artificial intelligence are both traditional areas of computer science. Model of a system *A* is usually defined as another system *B* which is somehow similar to *A*. By saying *modeling* we mean a process of expressing the model in form of some modeling formalism (programming or simulation language, for example). System *B* is always a simplified version of the system *A*, it contains just a part of its elements and their internal relations. We usually do not know the whole structure of the system *A*, we are not capable to cover the system *A* in its whole complexity or the system *A* is not realizable in our physical world.

The computer simulation is a process of *doing experiments* with the model. We gain a new knowledge of *A* during experimenting with its representation by system *B*. The term simulation is also understood as an execution of a simulation model (when speaking about models in programming languages). During the experimental phase, we have to confirm that *B* system is really similar to *A*. We validate the model, or debug or calibrate the model.

Both the game theory and computer science deals with the modeling of systems and their further analysis through the models. But there is a significant difference in doing that: the mathematical approach usually chosen by game theorists is called the *analytical modeling*, the computer approach is called the *simulation or computer modeling* (numerical). The analytical model in form of mathematical equations is a perfect conception of the studied system. We input the parameters to the equations and obtain the results immediately. The computer model must be put into a deep and time consuming experimenting. Moreover, as a numerical solution, it gives only approximate results. The experimenter has to keep in his mind, that he works with an imperfect and incomplete model, and the results are always infected by a certain error corresponding to the time spent in experimenting.

The analytical modeling seems to be the best way of doing the models. However, it is almost impossible to construct an analytical model of a larger system. It is practical to model the problems in analytical manner just in cases of very fundamental problems or in cases of some elementary parts of larger problems. For the rest of the tasks, it is highly recommended to build a computer model and to endure a time consuming experimentations. Modern computers already allow that.

The computer representation of a model always discretizes the studied problem. No matter if it is a meteorological model of atmosphere or a game-theoretical model of a decision problem, its domain (naturally continuous) is transferred to a finite enumeration – to a finite set. The cardinality of the discrete domain implies the number of computing operation necessary to proceed in the model execution. The computer

approach to model design is useful also for other reasons. The power of computers substitutes a human intellect spent in the analytical modeling. Contemplation of a mathematician can be transferred to a quanta of computing operations done by the machine.

The computer modeling has evident advantages in the field of modeling of complex systems, e.g. game situations. But we have to face some theoretical and practical computing constraints – the memory constraints and the time constraints. They both come from the algorithmic principle of the machine processing – searching the state (strategy) space. We have to keep in mind that every operation of the machine takes some time to be processed. The whole simulation may take hours, days or even years. Methods of Artificial Intelligence (AI) based on searching the state space use so called *heuristics*. The heuristics is an expert knowledge helping the AI algorithm to predict (observing the passed computation) what sub-spaces of the state space are not recommended for the further searching. Unfortunately, the heuristics are very connected with the particular model and they are not re-usable, they make the model over-complicated and decrease its flexibility.

We look for some universal heuristics which is not integrated in the application model. This approach should arise from some general game-theoretical law or a principle. Such a mechanism shall not require the modeler to influence his model with some application-specific heuristics. Such a mechanism shall experiment with an application model to find a faster way towards its solution.

We are going to introduce a framework suitable for development of such automatized mechanism and two examples of these mechanisms. It should be emphasized that the methods and algorithms described here were developed during many years of development of computer models of electro-energy markets. Case studies made by these models are requested by the government and commercial institutions in this sphere of industry.

The paper is organized as follows: the section 2 introduces and explains the basic assumptions and terms used in the paper. The section mentions a specific internal model *cellModel* evaluating the members of game state space. Let us imagine the *cellModel* as a procedure written is some programming imperative language. The *cellModel* takes the majority of the processor time and the whole methodology attempts to find a such algorithms which minimize the count of its invocations. The section 3 brings a more formal description of the game state space and its meaning in game-theoretical modeling. In the 4th section, the abstract architecture of a game model is presented. Its abstract parts will be concretized in sections 5 and 6, where an example of an equilibrium solver and example of a state space reduction mechanism are deposed. The last section 7 concludes the paper.

2. The modeling methodology

In market games, competitive situations are subject of modeling, in order to be able to better *understand the behavior of the players* in the real life or to be able to *predict the behavior* of real players (producers, traders and consumers). Theoretical literature on

gaming shows sometimes a certain measure of skepticism about whether the game theory can be successfully used for the prediction of future (a very interesting experiment is described in Green (2002)); in other cases, the usefulness of the game theory for predictions is defended (Erev et al. 2002). The game theory is undoubtedly a relatively successful and usable method. A number of papers (Krause et al. 2004, Yiqun et al. 2002, Gountis and Bakirtzis 2004, Kwang-Ho and Baldrick 2003) demonstrate that in the area of the electricity markets modeling (which is of interest of the author), mostly use the analytic models developing certain game-theoretical principles.

In this section, we introduce an approach of modeling the competitive situations. The whole methodology is heading towards a computer model able to predict the future evolution of the modeled systems.

We are specialized particularly in markets with centralized trading where all contracts are signed in the same time or the decision must remain constant for a certain time period (year). These situations are then modeled as strategic games – games in normal form (Myerson 2004). Let us first define a game in strategic form and some basic important terms.

Definition 1. A game in strategic form of N players is defined as:

$$\Gamma = (Q; S_1, S_2, ..., S_N; U_1, U_2, ..., U_N),$$

where

(i) $Q = \{1, 2, ..., N\}$ is a (finite) set of the players.

(ii) $S_i, i \in Q$ are finite sets of (pure) strategies of players i. Product of strategy sets makes the *game state space* $S = S_1 \times S_2 \times ... \times S_N$. Members of state space are called *strategic profiles* $s = (s_1, s_2, ..., s_N)$. Let S_{-i} denote similarly a subspace of S without S_i. S_{-i} notation will be frequently used to express a context of the i-th player's decision situation. Finally, $s_{-i} = (s_1, ..., s_{i-1}, s_{i+1}, ..., s_N)$ is a member of S_{-i}.

(iii) $U_i : S \to \mathbb{R}, i \in Q$ are utility functions assigning a payoff to each player i in each profile $s \in S$. We shall not delve too deeply into the analysis of the form of player's payoff; the payoff will simply be a number (often expressing player's financial profit). We expect the utility function not to be defined for some profiles. In that cases, utility function assigns some *game neutral payoff* (zero or a negative value). The utility functions U_i are usually implemented as $N-$dimensional arrays indexed by strategy profiles $s \in S$.

In some parts it will be preferred to denote shortly $\Gamma = (Q; S; U)$ to describe a game Γ with a state space S and utility functions U. We may write that the state space is *computed* to emphasize that the utility functions are known for all $s \in S$.

Modeling of games should lead to some equilibrium points. The strategic profile $s^* \in S$ consisting of the actions s_i^* made by individual players will be referred as a *game solution*. We expect the players to choose the best response on their opponent's possible action. The *equilibrium* is the mutually best response, which is formally defined in Definition 2.

Definition 2. Strategic profile s^* is the equilibrium in the game Γ, if:

$$\forall i \in Q, \forall s_i \in S_i, s_i \neq s_i^* : U_i(s_i^*, s_{-i}^*) \geq U_i(s_i, s_{-i}^*)$$

2.1 Strategy dominance

We have mentioned a need of some automatized mechanism that would experiment with the model to speed-up the classical searching in state space. The mechanism is supposed to reduce automatically the game state space following some general game-theoretical principle. The algorithms described here are based on examination of dominant and dominated strategies. Let us formulate the classical definition of the dominance.

Definition 3. Let us have a strategic form game $\Gamma = (Q; S; U)$. Strategy s_1^i of a player i strictly dominates strategy s_2^i, if

$$\forall s_{-i} \in S_{-i} : U_i(s_1^i, s_{-i}) > U_i(s_2^i, s_{-i})$$

Strategy s_1^i of a player i weakly dominates strategy s_2^i, if

$$\exists s_{-i} \in S_{-i} : U_i(s_1^i, s_{-i}) > U_i(s_2^i, s_{-i}) \wedge \forall s_{-i} \in S_{-i} : U_i(s_1^i, s_{-i}) \geq U_i(s_2^i, s_{-i})$$

The strategy dominance is connected to a technique called *iterative elimination of dominated strategies* which reduces the game state space. This technique is not trivial in the time-complexity point of view. On the other hand, we may handle the strategy dominance from the characteristics of the *Best-response*. The Best-response characteristics studies the player's behavior in situation when his opponents play strategies in sub-profile s_{-i}.

Definition 4. Strategies $S_{best}^i \subseteq S_i$ of a player i are his best-response in the strategy context $s_{-i} \in S_{-i}$, if:

$$\forall s_b \in S_{best}^i, \forall s \in S_i, s \neq s_b : U_i(s_b, s_{-i}) > U_i(s, s_{-i})$$

Set of Best-response strategies of the player i in the context $s_{-i} \in S_{-i}$ will be denoted as $BR_i(s_{-i})$, or simply $BR_i(s)$; $BR_i(s_{-i}, \Gamma)$ to emphasize the particular game Γ. Let us define a set of relevant (good) strategies $S_{BR,i}$ of a player i:

$$S_{BR,i} = \bigcup_{s_{-i} \in S_{-i}} BR_i(s_{-i}) \tag{1}$$

Obviously, if $\forall s_1, s_2 \in S_i : U_i(s_1, s_{-i}) = U_i(s_2, s_{-i})$ holds for some $s_{-i} \in S_{-i}$, the best-response is equal to S_i. We will consider this state to be *not suitable* for the game analysis. It is preferable to work with such games, where $\forall s_{-i} \in S_{-i} : |BR_i(s_{-i})| \ll |S_i|$. Anyway, it is ideal if $|BR_i(s_{-i})| = 1$.

With the framework of strategy dominance, these statements can be considered:

(i) A rational player never plays a strategy, which is (strictly or weakly) dominated by another strategy (see G-solve in section 5.4).

(ii) A rational player always plays a strategy from $BR_i(s_{-i})$ in the sub-profile s_{-i}. The fact of $|BR_i(s_{-i})| > 1$ introduces a certain uncertainty and hence a mixed behavior.

(iii) A player always decides within the set $S_{BR,i}$. The strategies $S_i \setminus S_{BR,i}$ have no meaning for him. We are going to call them simply the *dominated strategies*. The strategies $S_{BR,i}$ will be called the *dominant strategies*.

Distinguishing the dominant and the dominated is the starting point for proxessing the game tasks. Splitting the strategies to dominant and dominated is not evident directly from the Γ game specification. To analyze the dominance, the algorithm must go through the whole state space. This is impossible in many practical cases for the complexity reasons (see section 2.3). In this paper, we are going to demonstrate that it is realizable to analyze a relevant sub-part of the Γ state space to compute the best-response characteristics of the players. Concentrating only to $S_{BR,i}$ strategies (as a minimal representative of the original Γ) the computer can find the game solution of Γ without analyzing the whole state space.

2.2 Strategic equivalence of games

Strategic equivalence of games (Myerson 2004) is a wider term. We introduce a specific definition of the term relevant for this paper.

Definition 5. A game $\Gamma_r = (Q; S_r; U_r)$ is strategically equivalent to another game $\Gamma = (Q; S; U)$, if holds:

(i) $S_r \subseteq S$ and

(ii) $\forall s_r \in S_r, \forall i \in Q : U_{r,i}(s_r) = U_i(s_r)$ and

(iii) $\forall i \in Q : \bigcup_{s_{-i} \in S_{-i}} BR_i(s_{-i}, \Gamma) = \bigcup_{s_{r,-i} \in S_{r,-i}} BR_i(s_{r,-i}, \Gamma_r)$.

This means that a game Γ_r is strategically equivalent to Γ if it is played in the same set of players, just the state space S_r is a subset of S and no strategically important (dominant) strategies are omitted.

The task is to find a strategically equivalent game Γ_r for a given game Γ using an efficient algorithm, i.e. using an algorithm with better time and memory complexity than the method of iterative elimination of dominated strategies (or its equivalent in the meaning of best-response characteristics). More reasoning for that is given in the following section.

2.3 Algorithmic complexity of the game analysis

The time and memory algorithmic complexity is the essential problem in area of these models. We are not going to study rigorously a complexity of the algorithms, neither from the point of view of game theory nor from computer science point of view. This particular research is connected to the real existing models. The algorithms and

methods are developed to allow their user to proceed his practical experiments in reasonable time using the available computer machines. Thus, we simply differentiate between the good and bad complexity. We keep in mind the words of Kamal Jain: "If your PC cannot find it, then neither can the market." As the market can find *it*, we have to make a model which finds it as well. This sort of disproportion between the complexity of the reality and its model is common for all AI algorithms.

The time complexity of an algorithm is given by number of computing operations necessary for computing the response for a given input. In the area of real computers, this is the time we have to wait for the result.

The memory (space) complexity is given by number of memory cells necessary to finish the computation for a given input. The real computers are limited by their physical operation memory. In the game theoretical models specifically, we are often limited by the fact that the whole N-dimensional matrix of utility functions U can not be present in the computer memory.

Which one of these two complexity problems is the one stopping us to complete the task? Practically, we do not care about the time complexity of the analytical algorithm solving for example the strategy dominance. Mostly, we have to care about the input game state space which does not fit the computer memory. Let us demonstrate that on the following example.

Example of the algorithmic complexity

Let us assume a rather simple algorithm (Algorithm 1), which computes the *Best-response* characteristics for a given player i and a given context s_{-i}. The Algorithm 1 terminates with a set of $BR_i(s_{-i})$ strategies.

Algorithm 1 Best-response algorithm

```
Input: sub-profile s_-i, player i
max := -MAXIMUM_NUMBER
list := []
for s_i in S_i:
      u := U_i(s_i, s_-i); Comment: If U_i(s) is not known, it must be computed by
cellModel(s) first – this is the main computational load in the whole algorithm.
   if (u=max):
      add s_i to list
   else:
      if (u>max):
         list := [s_i]; max := u
return list;
```

Now, let us consider a 8-player game where each player has 100 strategies, so $Q = \{1,2,...,8\}$ and $|S_i| = 100\ \forall i \in Q$. The size of the whole state space is $|S| = 100^8 = 10^{16}$ cells. Each cell $s \in S$ keeps an utility $(U_1(s), U_2(s), ..., U_8(s))$ – a vector of eight numbers. Let a number takes 4 bytes of computer memory, so the cell of eight utilities takes 32 bytes in total.

Computing $S_{BR,i}$ in (1) using Algorithm 1 requires the utility function U_i to be known for all $s \in S$. At first, let us assume that the whole $U_i(s)$ is stored in the computer memory. Then, it requires $10^{16} \cdot 32$ bytes of memory which is $32 \cdot 10^{16}$ bytes, or approximately $32 \cdot 10^7$ Gigabytes (GB) of RAM memory (the current PC computers have usually 1-8 GB RAM). This whole eight-dimensional matrix of eight-dimensional vectors clearly can not be stored in memory of any existing computer.

Secondly, let us assume that $U_i(s)$ is not stored in memory, but there is a function (a computer procedure) computing the *i*-th player's utility for any $s \in S$ (this computer procedure will be called *cellModel* in this paper). Such a function is computed in a physically existing computer and it takes certain time – let us say $1\mu s = 10^{-6}s$ (using a standard PC processor doing approximately 500 million floating-point operations per second). The computer thus can compute one million requests for the utility function $cellModel_i(s)$ per second. We will use the transcription $U(s)$ to highlight that the value of the utility for s is known and the transcription $cellModel(s)$ (or $cellModel_i(s)$) to emphasize the need to compute (as a computer program) the value first.

Let us recall the Algorithm 1. Computing $BR_i(s_{-i})$ for a given $i \in Q$ and $s_{-i} \in S_{-i}$ requires the $cellModel_i(s)$ function to be invoked $|S_i|$ times (i.e. 100 times in our example). There is surely some computing overhead in the Algorithm 1 it-selves but we neglect it. To compute all $S_{BR,i}$ for all $i \in Q$ we need to invoke $cellModel_i(s)$ $[\Pi_{i \in Q}|S_i| \cdot |S_{-i}|] = |Q| \cdot |S| = 8 \cdot 10^{16}$ times. Doing one million iterations of $cellModel_i(s)$ per second, we need $8 \cdot 10^{16} \cdot 10^{-6}$ seconds to get know all $S_{BR,i}$, which means about 2500 years.

We can see, that the game-theoretical algorithms analyzing the games work only in cases of very small games.

2.4 Designing a game-theoretical numerical model

The game theory usually regards Γ (see an example in Figure 1) according to the Definition 1 as the *problem specification* (i.e. including the assignment U_i). The game theory is then a package of analytic tools helping us to find the solution out of the set of players, their strategies and the numerical analysis of their payoffs (Osborne and Rubinstein 1994).

Figure 1. An example of a game Γ

	left	right
top	10,2	8,4
bottom	5,4	7,3

In this paper, we deal with modeling of problems implementing the game theory and its algorithmization (so called algorithmic game theory Nisan et al. (2007)). The specification of Γ is not regarded as an input for our deliberations but rather as an interim result of our computations. Searching for the final prediction in form of the

equilibrium is just a final step of the long process which passes through the following phases (i)-(vi). First ideas of this methodology were published in Hrubý and Toufar (2006).

(i) *Specification of an abstract model of the situation* – the main objective of the model and its known and unknown parts must be specified. In the case of market models, we specify the purpose of the model, the market commodities under the study, time range of the forecasting and others.

(ii) *Identification of players and their mutual relations* – mapping of real institutions (e.g. producers and buyers) to strategic game players is specified. This is followed by the modeling of their economic potential, their knowledge of the system and key features of their strategic behavior (ideal competition, oligopoly, leader-follower,...). For example, all regional traders with electricity are aggregated to a single national buyer. Let us note that at the level of a large international-scale business operation (as e.g. long-term contracts for the delivery of electricity) we assume *complete information* about the game among the players (game structure including utility functions is *common knowledge* to all players).

(iii) *Game rules determination and selection of the type of strategic equilibria* – in the sense of the game theory we differentiate between two basic game classes: strategic and sequential. In most cases, our games are strategic (games in strategic form, normal-form games, matrix games). There are variants of equilibria including Nash equilibrium (Nash 1951), Stackelberg equilibrium (Latorre and Granville 2003), correlated equilibrium (Aumann 1974, Papadimitriou 2005) and others. The games also contain nested (inner) strategic sub-decisions in form of nested games or auctions (Krishna 2002).

(iv) *Model of the game strategy and strategy generation* – we work with numerical models and, therefore, with a certain degree of discretization of unknowns. Moreover, in real game situations, we introduce various forms of hierarchical decision-making with multi-dimensional strategies. For example, let us consider the decision of a producer to supply his product to various national markets (geographically) or to allocate his production capability to different particular products. In the electricity case, we differentiate between various time-defined commodities (year base supply, monthly base supply, peak-load,...) or system reserves. The producer decides about to sell his production to his domestic customers or to export it abroad.

(v) *Game payoff determination model* – this part represents the expert core of the whole issue. The objective is to design and to implement the model of the situation specified by the given strategic profile. To put it in other words, we are developing an application-specific function $cellModel(s,C)$. This function returns the players' utility vector for the given profile $s \in S$ and the global game context C. It is a modeling and computational implementation of functions $U_i : S \to \mathbb{R}$.

There is a significant difference between the utility functions U and the function *cellModel*. From the Computer science point of view, U_i is a mathematical function returning the player's payoff instantly (in zero time); U_i is a memory record, an array indexed by strategy profiles $s \in S$. On the other hand, this information must be computed first (by the *cellModel*), this computing load takes some processor time and makes the simulation experiment time consuming. Resulting value of *cellModel* is then stored in U. A similar approach was published in Viguier et al. (2006). The whole state space is computed in the loop as it is shown in Algorithm 2. Time complexity of this computation may be very high, as the upper border of the complexity is defined by the number of iterations $|S|$ and by the complexity of *cellModel*. Indeed, we do try to establish theoretical principles which will allow a reduction of the computational load (Hrubý and Čambala 2008). That is the sense of this paper.

(vi) *Theoretical analysis of the game and the determination of the equilibrium* – once all information describing the game according to the definition Γ is complete (all U_i are computed), we need to algorithmically determine the rational behavior of players within the game. The result of the game is thus the profile $s^* \in S$, which we compute (according to the particular equilibrium concept) as the most probable strategic decision of players participating in the game. This phase is enormously demanding computationally too. The complexity can be reduced by the analysis of the state space and by the elimination of strategies, which a rationally thinking player would not be likely to play.

It should be emphasized very strongly, that the so called internal model *cellModel* computing the payoff vector for each strategy profile $s \in S$ is not just a simple function *revenues* $-$ *costs*. The *cellModel* for certain $s \in S$ is supposed to be a rather complex model of what all happen in the real life if players $i \in Q$ adopt their strategies s_i. Regarding the particular application, *cellModel* may model their strategic offer to various markets, their behavior in inner sub-decision making, their bids in auctions and their production model. If we say *revenues* $-$ *costs*, some mechanism must compute the level of the revenues and some other mechanism must specify the costs. In our particular case, computing the costs is very complex as it requires an optimization of the production process. For example, in the models with one-hour time granularity, the *cellModel* procedure computes an optimization of working point of all included power plants (linear programming task) for 8760 hours of the year. This explains why *cellModel* can not be expressed analytically as a mathematical function but implemented as a numerical model and the whole application model works in discrete strategies.

Example of a numerical model

Successive phases of the game-theoretical modeling can be best shown in a fictitious oligopoly example. Let us start from a situation involving a group of N players, where each player i is fully defined by his production machines $M_i = \{m_1^i, m_2^i, \ldots\}$. Operation of the machines is simplified to these three parameters:

(i) *fixc*: $M \rightarrow \mathbb{R}$; fixed costs of the machine which must by paid no matter if the machine produces or not (it is mostly an amortization of the machine).

(ii) *prodc*: $M \rightarrow \mathbb{R}$; production costs per unit of production (e.g. coal, gas,...).

(iii) *cap*: $M \rightarrow \mathbb{R}$; production capacity of the machine in number of units.

The demand for the (single) commodity is considered to be invariably *demand* (which is a usual situation at energy markets). In terms of the oligopoly price gaming we ask a question: What are the price and volume of production that individual players will successfully achieve at the market? We assume that players know each others production parameters. The game is regarded as a strategic form game, i.e. all players $i \in Q$ choose their strategies s_i^* concurrently, at the same time, and the result of the total selected profile $s^* = (s_1^*, s_2^*, ..., s_N^*)$ are payoffs $U_i(s^*)$. As for the equilibrium concept, Nash equilibrium is selected. In this manner we pass phases (i)-(iii).

Furthermore, we have to specify sets of strategies S_i of individual players and subsequently to design the game internal model *cellModel* computing payoffs $U_i(s)$ in profiles $s \in S$. We treat the game as a discrete system, and, therefore, we also would like to have *discrete* sets of strategies. The searched quantity, i.e. *commodity price*, is expected to be in the interval $p = \langle p_f, p_t \rangle$ ($p_f, p_t \in \mathbb{R}$). When modeling a real commodity market, we indeed expect that the resulting price will lie in a certain reasonable range and, therefore, the finiteness of p is acceptable. The price range can be converted to a discrete form in an arbitrary manner, e.g. in the form of a regular series $Prices = p_f, p_f + step, p_f + 2 \cdot step, ..., p_t$. We may set $S_i = Prices$ for all players $i \in Q$; generally, however, sets of players' strategies can differ for each individual player. The main problem in this approach consists in the correct determination of the discretization step $step$ which also defines the precision of the prediction model. The correct construction of the set of strategies is a separate expert problem which will be analyzed later on.

The determination of S_i yields the state space $S = S_1 \times S_2 \times ... \times S_N$. In order to evaluate the payoff of players in the game, we have to invoke a certain internal model $cellModel(s, C)$ for all $s \in S$ and a certain global context. In our case, the context is given by $C = \{demand\}$. The $cellModel((s_1, s_2, ..., s_N), C)$ is a procedure computing what offers are the players going to make at s_i price, how they will succeed in the competition and what profit they can make in the profile s (see Hrubý and Toufar (2006) for more details). The following procedure $cellModel(s, C)$ models what happen in one strategic profile $s \in S$. The *cellModel* procedure is a sequence of the offer, trading and production settlement.

Let us say that every player i offers the production capacity of those machines that are profitable in the context of his price strategy s_i; $s \in S$:

$$M_i^p(s_i) = \{m \in M_i | prodc(m) + fixc(m) \geq s_i\}$$

$$offer_i(s_i) = \sum_{m \in M_i^p(s_i)} cap(m)$$

The total offer of all players $i \in Q$ is then:

$$totalOffer(s) = \sum_{i \in Q} offer_i(s_i)$$

The buyer receives $|Q|$ offers in form $bid_i = (i, amount_i, s_i)$ where s_i represents the price required by the player i and $amount_i$ is the number of units of the modeled commodity. The buyer wants to buy *demand* units of the commodity (we assume no price-demand elasticity). Let *Bids* is a list of bids ordered by price (supply curve).

By trading, we mean a transformation of the list *Bids* to a similar list $Sells = \{(i, sell_i, s_i), ..\}$ of amounts $sell_i$ accepted by the buyer from each producer i for price s_i. In the profile s, if *totalOffer(s)*$\leq$ *demand*, all bids are accepted, so $sell_i = amount_i$ for all $i \in Q$. Otherwise, the buyer buys up to *demand* units respecting the offered price.

The producer receives the information about the contract made (amount $sell_i$ sold for s_i price). He arrange his production set to produce $sell_i$ for optimal cost – let $Y_i(sell_i)$ is a optimizing procedure spreading the $sell_i$ load over his set of machines M_i. The result of the optimization is a list of tuples (*machine, production*) where (2) holds.

$$sell_i = \sum_{(m_j, p_j) \in Y_i(sell_i)} p_j \tag{2}$$

$$optimalProductionCost(sell_i) = \sum_{(m_j, p_j) \in Y_i(sell_i)} prodc(m_j) \cdot p_j$$

$$profit_i(sell_i, s_i) = sell_i \cdot s_i - optimalProductionCost(sell_i) - \sum_{m \in M_i} fixc(m)$$

To complete the procedure, the $cellModel(s, C)$ is a sequence of steps:

(i) Collect bids from all players.

(ii) Intersect the supply curve with the *demand* and generate messages $sell_i$ to the players.

(iii) Players optimize their production units regarding the accepted amounts $sell_i$ and enumerate their outcome
$profit_i(sell_i, s_i)$.

(iv) $cellModel(s, C) := (profit_1, profit_2, ..., profit_N)$

This $cellModel(s, C)$ is invoked for all $s \in S$ (see Algorithm 2). After $|S|$ iterations, all information of the game is evaluated, and we can proceed toward the analysis of the game to determine the expected forecast – the equilibrium. Implementing the model would be trivial for anyone. We just wanted to demonstrate the functionality of the so called *internal model* computing the payoff of all players $i \in Q$ acting in the s strategic profile and the global context C – $cellModel(s, C)$.

Let us remind the *heuristics* and the preference. Can the player i generally know whether he makes better (selling more for a higher price) when playing s_1^i rather than

s_2^i in the context s_{-i}? Can the heuristics during computation of $cellModel((s_1^i, s_{-i}), C)$ easily predict the player's contract in the profile (s_2^i, s_{-i})? No, the player (all players) must *try that* invoking *cellModel* for both profiles. All attempts to implement heuristics into *cellModel* which, having a knowledge of the value of $cellModel(s_1)$, shall predict the next outcome in s_2 profile will fail. Moreover, such a functionality makes the *cellModel* procedure more complicated and restrain to the further modification of the internal model. For this reason, we look for an automatized heuristics which does not need to see inside the internal model.

2.5 The status of game equilibrium

The equilibrium is understood to be a rule, or a model specifying a probable behavior of players in the game context – it means, within the strategies and known payoff functions. In our models, we strictly assume that the utility functions and other parameters are *common knowledge*. This paper also shows that when using our methodology, choosing a particular equilibrium concept is just a final computing operation over the game state space.

The time and space complexity is very important for the construction of these models and computing the equilibrium in the simulation experiments. We differentiate between the equilibrium algorithms where we expect or do not expect the U to be computed:

- The U is expected to be computed. We just analyze the computed state space to find the equilibrium points (traditional algorithms computing Nash equilibrium).

- The U is not expected to be computed. The algorithm starts from zero knowledge of U and touches (*cellModel*) just that cells which the algorithm needs for its operation. Putting the *cellModel* computation and the equilibrium determination together may decrease the number of accessed cells from S (in compare to the previous concept). This may decrease the computing time to a fraction of the conventional approach (G-solve).

2.6 Two-level approach to modeling of game situations

An efficient computer implementation of game model was the opening idea for the two-level model architecture. The efficiency means the computing efficiency mainly. The *software engineering* is the second view to the efficiency. We also require the model (as a software work) to be easy to understand and flexible for future modifications.

The traditional AI algorithms work with the state space, evaluate its cells in form of some utility and search for the optimum (methods of searching in the state space, problems solving, playing games). All these methods employ various forms of heuristics to decrease the computing complexity. The heuristics are pieces of programs containing a particular expert knowledge of the problem to be solved. They help the AI algorithm to predict that parts of the state space which are useless for the analysis and redundant for

the search of the optimum. From the modeling (software engineering) point of view, the heuristics are parts of the model badly influencing its clarity and flexibility.[1]

There is a compromise and rather good solution: computing of the utility of each particular cell (*cellModel*) is separated from the mechanism searching in the state space. We say that the mechanism searching in the state space experiments with the model $cellModel_\Gamma$ of a given game Γ. The basic mechanism is shown in Algorithm 2.

Algorithm 2 Basic search in state space

```
for s in S:
  U[s] = cellModel(s,C)

eq = determineEquilibriumPoint(S,U)
```

The Algorithm 2 goes through the whole state space and enumerates $U(s)$ for all $s \in S$. The time complexity of Algorithm 2 is given by the cardinality of S and the programming implementation of *cellModel*. The computing complexity of *cellModel* is not trivial as the *cellModel* may contain other sub-decisions (inner games), production optimization and so on. Moreover, we do not expect that the whole contents of S can fit the computer memory. The Algorithm 2 is thus just a *theoretical demonstration*. Its time and memory complexity makes the theoretical upper boundary of the state space computation. It is the worst solution and we attempt to find the better ones. An algorithm verifying that a given profile s^* is or is not the equilibrium, is the lower boundary.

We expect that the *mechanism* processes the state space of Γ in such a way which is efficient from the point of view the particular game, and type of the chosen equilibria concept. The algorithm searching the state space of Γ shall terminate (output) with:

(i) A minimized game $\Gamma_r = (Q; S_r; U_r)$ without dominated strategies $(S_i \setminus S_{BR,i})$ which is strategically equivalent to Γ. The equilibrium of Γ is then computed using some other procedure (for example using some already implemented solver like GAMBIT (Gambit homepage 2008).

(ii) Solution of Γ in form of a game equilibrium s^* (analyzed using Γ or Γ_r).

We are going to present two different approaches in this paper. The first one demonstrates an efficient computing of the equilibria (the correlated equilibrium (Aumann 1974) in our case). The second one searches the state space in extremely huge games to reduce the given game into its strategically equivalent version without dominated strategies. Both methods are based on analysis of strategy dominance.

[1] There are some other very efficient methods of dynamic reduction of strategy sets. However, they exceed the topic of this paper.

3. State space of the strategic games

We model the decision processes. An intelligent entity is supposed to make a decision and therefore, it analyzes its *options*. A strategy is taken in this context as a possibility to act somehow. The entity thus collects all its options and starts to evaluate them. We know, that in the strategic games, the entity evaluates the strategies in context of other players.

We follow the model design and experimenter's point of view. It is then necessary to include this important fact: *Construction of a representative and reasonable strategy sets is an inseparable phase of the model design*. The artificial intelligence yet has not advanced so far that the AI algorithms would be able to pass that by their own. To put it in other words, we do not expect the AI algorithm (or the prediction model as a whole) to specify itself a task. Specification of the task is still left on the experimenter (user).

3.1 Generating the strategy sets

Let us assume that the model and all the computations will be held in discrete state space. We may understand the modeled commodities to be continuous in their basic principle (price, quantity). In the real life, we treat them as discrete anyway (elements of currency (cents), production attributes of the machines, satisfying the quotas and rules,...). We just have to specify the way of making these things discrete. The quantities related to manufacturing are rather clear (production units, size of a package and so on). The meaning of commodity price is the only trouble.

The continuous interval $P = \langle p_f, p_t \rangle$, within the price decision is expected, can be easily discretized by a sampling period $step$. It makes a regular sequence $S_i = \{s_{i,1}, s_{i,2}, ...\} = s_f, s_f + step, ..., s_t$. The size of $step$ is essential here. We lose an information value and sense of the model when a too long $step$ ($step$ is a big number) is used. When having a too short $step$ (a small number), one would assume that the accuracy of the model gets increased (no matter if it increases the computational load). However, the too short $step$ just influence the model with a *mixed behavior*.

Let us have an example. The price of a commodity is expected to be withing $P = \langle 0, 10 \rangle$. Setting the $step = 1$ makes the player distinguishing his optimal price $s^*_{i,x}$ (for example let $s^*_{i,x} = 7$) very clearly. On the other hand, when operating at $step = 0.1$, after a significantly longer computation, we achieve a probability distribution over the pure strategies like $0.2 \cdot s_{i,x-1}, 0.7 \cdot s_{i,x}, 0.1 \cdot s_{i,x+1}$ (price strategies $6.9, 7, 7.1$). Can be such a result accepted as *the better solution*? Does the model operating on $step = 0.1$ a better job than the model with $step = 1$? When doing predictions of the commodity price in the far future (e.g. year 2025), does anybody believe the model to predict the price with 0.1 precision?

3.2 Differentiation of strategies

We have described choosing of the right (reasonable) discretization step and its influence to complexity of making the decision. A rational player must always be able to

show (to satisfy the definition of the rationality) for any two his actions s_1, s_2, if s_1 is preferred to s_2 or vice versa, or that both strategies are equal in their significance. We compare the strategies using their corresponding utility.

Definition 6. A set of strategies S_i of a player i is *distinguishable* in sub-profile $s_{-i} \in S_{-i}$, if

$$\forall s_1^i, s_2^i \in S_i, s_1^i \neq s_2^i : U_i(s_1^i, s_{-i}) \neq U_i(s_2^i, s_{-i}).$$

Similarly, we define that the strategies $s_1^i, s_2^i \in S_i$ of a player i are distinguishable, if

$$\forall s_{-i} \in S_{-i} : U_i(s_1^i, s_{-i}) \neq U_i(s_2^i, s_{-i}). \tag{3}$$

Differentiation of strategies is useful for analysis of the game state space. It also forbids the mixed behavior in the game. A player can clearly (purely) see his good and bad strategies. We say that the strategy set S_i of a player i is *well distinguishable*, if the previous condition (3) holds for all $s_1^i, s_2^i \in S_i$. This condition is perhaps too strong and very rare in the real world. Moreover, the dominance of strategies is never obvious when the utility functions are not available (it is not directly clear that a player gains more for price strategy €30 than for price strategy €25). Let us express the complementary view on the distinguishability in the form of the Best-response characteristics. The state space is well distinguishable if

$$\forall i \in Q, \forall s_{-i} \in S_{-i} : |BR_i(s_{-i})| = 1, \tag{4}$$

or the number of best-response strategies for some sub-profiles remains small in compare with the size of the strategy sets. Such a setting will not cause a state space explosion during the game analysis and the whole computation will remain in reasonable response time.

Such a situation can be achieved if the *cellModel* model is designed with a special care about enumerating the final payoff in the profiles. Let us demonstrate the situation using a game where the players choose their strategy in form of tuples $(price, amount)$. We will call that the *multi-dimensional decision*. Let us imagine a player who succeeded to sell the same amount *sell* when playing strategies $s_1^i = (price, amount_1)$ and $s_2^i = (price, amount_2)$ in the same context s_{-i}, but mainly $amount_2 > amount_1$. That is an extension of the example in section 2.4. His payoff for the sold amount *sell* and strategy $(price, amount)$ is (the production costs are ignored):

$$U_i(s_j^i) = profit_i(s_j^i, sell) = sell \cdot price$$

The strategies s_1^i and s_2^i are indistinguishable as $U_i(s_1^i) = U_i(s_2^i)$ holds. It is rather evident that a real player would prefer the strategy s_1^i because s_1^i reserves a smaller production capacity. If we include this reservation of the installed production capacity to the equation on the side of production costs ($r \in \mathbb{R}^+$ is the cost of the capacity reservation or lost income per unit), we achieve the wanted distinguishability:

$$U_i(s_j^i) = profit_i(s_j^i, sell) = sell \cdot price - amount \cdot r$$

Let us conclude, that the situation when $U_i(s_1^i, s_{-i}) = U_i(s_2^i, s_{-i})$ expresses that both strategies s_1^i and s_2^i are equally preferred by the player i. This state must have some *reason* and motivation. It may become, that two different contracts (in their structure) give the player an equal profit, but this is very rare in our experience. It is mostly an inability of a player to make a decision or an inability of a modeler to evaluate the contracts properly.

3.3 Multi-dimensional strategies in decision making

In the preceding section, we briefly introduced the modeling of multi-dimensional decisions, i.e. a situation when a player makes a decision within the framework of strategies generated in the form of Cartesian products of more quantities. In the previous example, the player has been choosing the strategies in form $(price, amount)$. In our decision models, the players make decisions composed from other sub-decisions. Multi-dimensional decisions model the situations with more traded commodities or more possible markets to make the contracts. The players have to decide about spreading their production capacity to more commodities, they have to decide the price for each particular commodity and the particular market to place it.

If the price is supposed to be within the interval $Prices = \langle p_f, p_t \rangle$ and the amount within $Offer = \langle 0, cap \rangle$, then the strategy set of the player i is given by $S_i = Prices \times Offer$ (and discretized somehow again). We are not going to provide a complete methodology of modeling multi-dimensional decisions composed of elementary sub-decisions. More can be found in Hrubý (2007).

We know that the state space S of the game Γ is given by $S = \Pi_{i \in Q} S_i$. The state space represents an exhaustive listing of variants, that may occur in the game. Nash (1951) proved that *each finite game has at least one Nash equilibrium* (generally in mixed strategies). This means that, in any finite state space S, we may find a stochastic outcome $s^* \in S$ that will satisfy the definition of the Nash equilibrium. If S is the input for the computational model, the model will always point to a solution. For that reason, the design of S_i is of key importance. A false specification of the strategy of players may tilt the result so that it departs from the real situation.

The composition of the strategy sets of individual players is scalable and thus well suited for experimentation. If $|S| = 1$, then the solution is clear and there will be no game at all. The classic game theory usually assumes that each player i has at least two strategies so as to be the *strategic player*. In our concept, a player belongs to the game as long as his impact is included in the internal model *cellModel* regardless of the number of his strategies. Player i with $S_i = \{s_1^i\}$ is the *participating player with constant behavior* s_1^i.

A similar situation arises in the pattern of multi-dimensional strategies. If the player i is modeled in such a manner that he has a set of D_i elementary decision-making problems, where each elementary problem $d_j^i \in D_i$ belongs to the finite non-empty listing $Dbase(d_j^i)$, the set of strategies S_i of a player is again given as (5).

$$S_i = \Pi_{d_j^i \in D_i} Dbase(d_j^i) \tag{5}$$

From the algorithmic point of view, the multi-dimensional strategy (a vector) is fundamentally equal to a scalar-type strategy. The whole methodology of two-level modeling and automatized mechanisms searching in the state space remains unchanged. Multi-dimensionality of strategies just extends the size of strategy sets. But significantly.

4. Implementing the model

The purpose of the modeling in this area is to design the parameters of a game following the specification in section 2.4, to identify particular decisions D_i of players and their domains $Dbase(d), d \in D_i$ (section 3.3). Proper designing of *cellModel* function is an individual expert problem. It is rather important that the *cellModel* well differentiates between the player's strategies. This will minimize the mixed behavior in the game which makes the computing unnecessarily more complicated (and the result is not better in our opinion). The mixed behavior must have some reason in the game.

Having *an unique* pure equilibria is the preferable end of the analysis. Having just a single equilibrium eliminates the need of further interpreting the results. Otherwise we have to decide (algorithmically) what equilibrium is the most authentic as a behavior of the players.

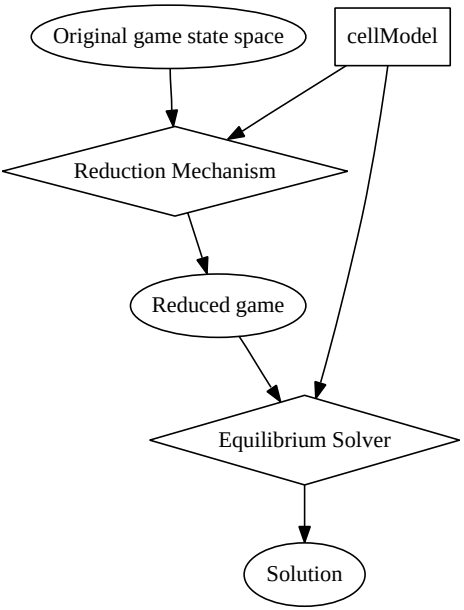
The described methodology shows a possible computer implementation in form of a programming library. The application model can be then constructed from the predefined program elements. We develop a C++ library called GameLIB to support rapid development of the application models. There is another example of a rather general game library called GAMBIT (Gambit homepage 2008). The GAMBIT library contains just the basic equilibrium algorithms and is not very suitable for this type of modeling.

4.1 An example (MCE)

In one of our models – Model of Central Europe (MCE) – we model a strategic decision of eight Central-European electricity producers in the region. They have to negotiate year and month contracts of electricity delivery. We model all their power stations and their detail operation including their time-variable production availability, fuel consumption, interconnection to other industrial use etc. The decision on contracts (amounts, prices) for the main yearly contract and twelve monthly contracts must be done in one moment. The strategy (as one possible action of behavior) is thus a large vector including many sub-decisions like: amount of yearly production to offer at the domestic market, amount to export in yearly contracts, amount to reserve for monthly contracts, required price for the commodity, and the same decisions for monthly contracts. Moreover, the model includes another eight strategic buyers who decide how to purchase their demand effectively (from the domestic producer or from the import, their price-demand elasticity and the value added based on electricity consumption).

The state space S has then sixteen dimensions (eight producers and eight buyers). Every strategic profile $s \in S$ consisting of players' multi-dimensional strategies $s_i = (s_1^i, s_2^i, s_3^i, s_4^i, ...)$ predestinates their behavior in the profile s: the player-producer i offers s_1^i amount to its domestic market for s_4^i price, s_2^i offers to the export (which

Figure 2. Basic composition of computation (Top view to the model)



requires an inner decision about the particular markets to export and about the strategic bids to offer in auction for electricity transmission lines) and reserves s_3^i for the twelve monthly contracting. The yearly contracts are made in these eight markets and the similar process is repeated twelve times to make the monthly contracts. Buyers strategically decides about the splitting their demand between the yearly, monthly contracts and various electricity producers to achieve the best prices. The *cellModel* again implements all the activities regarding the processing the particular profile s.

This paper does not study a particular model, it describes some of the methods developed together with our models. Detail description of MCE (or other model) would be a topic for another large paper.

4.2 Top-level of computation

Computing the prediction of players' behavior in a given game is well scalable by specification of its state space. The computation is very dependent on the particular application, especially on its internal model of behavior which we call *cellModel* here. We proceed the practical experiments using a machine having 8xCPUs Xeon 2.66 GHz and 16 GB of the RAM memory (or alternatively on 4xCPUs AMD Opteron 2.8GHz with 32 GB RAM). The computation may take minutes, hours or the time exceeding our requirements.

The general structure of the whole experiments passes two basic phases (see Fi-

gure 2). The initial game state space (Original game) has obviously a plenty of redundant dominated strategies which the rational player is never going to play. The state space has an extreme cardinality. This is the task which the experimenter formulated for the simulation. Usually, it contains all possible and hypothetical strategies to ensure that no behavior was forgot. It is not possible to analyze the state space in this form (see section 2.3 for a brief demonstration of the algorithmic complexity). A reduction mechanism is invoked to predict subsets of good strategies (for example there might be tens of them for each player).

The equilibrium solver is then responsible for determination of the equilibria. In this two-level game modeling, the expert core (*cellModel*) and the rather general experimenting mechanism (reduction mechanism, equilibrium solver mechanism) are separated. Let us conclude the overall motivation for this approach:

(i) We have to construct a practical computer model capable of certain analysis and prediction.

(ii) The model must be enormously flexible and ready to accept any modification of its structure, specification and mission.

(iii) The experimenter may not be limited in specifying his queries (state spaces).

(iv) Model execution must be maximally efficient and fast responding. We assume the experimenter doing large batches of experiments.

(v) The model (and the simulation method) must be ready for parallel processing.

The following two sections give some response to these requirements.

5. Computing the correlated equilibria

Correlated equilibrium (CE) (Aumann 1974, Papadimitriou 2005) is a well know game theoretic concept extending the classical Nash equilibrium with a special synchronization device helping the players to make their decision. A rational player then agree that incoming event (signal) recommends him the best strategy do choose. This is an opposite to the Nash equilibrium (NE), which assumes no communication platform between players and their surrounding environment. The players then prefer to make careful actions, often leading to lower common social outcome and misunderstandings. Following our experience and results, we do believe that a rational player in market competition (where the rationality is a common knowledge) behaves in the manner of correlated equilibrium. More reasoning for the use of the correlated equilibria has been done in Samuelson (2004).

Correlated equilibrium is computable in polynomial time (Papadimitriou 2005) as a linear programming (LP) task maximizing the common outcome of all players in context of game constraints. Unfortunately, in our practical games with large number of players and their strategies, the task is still too huge to be solvable on standard PC computers. On the other hand, most of the games coming from real situations can

be converted to their minimized strategic equivalents (where *dominated strategies* are eliminated) which are computable in a fraction of the original computing time. And we would like to algorithmize this transformation.

Let us assume a previously defined N-player game Γ with already computed strategy state space S, so that all U_i are known for all $i \in Q$ and $s \in S$. Let us assume that all strategy profiles are ordered $(s_1, s_2, ..., s_{|S|})$ and can be indexed by an integer $j \in \{1, ..., |S|\}$. By computing the correlated equilibria we obtain a row vector $p = (p_1, p_2, ..., p_{|S|})$ of probabilities assigned to all strategy profiles (we also say, a probability distribution on strategy space). CE is a form of Nash equilibria in *mixed strategies* (MNE) and similarly like in Nash equilibria, more than one equilibrium point can satisfy definition of CE. Regarding the presumption of rationality among players we search for a unique Pareto efficient CE. The following technique gives a profile with maximum payoff for all players where no one wants to deviate.

5.1 Basic approach

Correlated equilibrium is computable as a linear programming (LP) problem where we maximize the global objective function Z in (6) with probability variables p_j satisfying (7) to obtain the best solution for all players together (Pareto optimal solution). The LP-task in (6) and (7) is bounded by linear constraints (8).

$$Z = \sum_{j=1}^{|S|} p_j Z_j \tag{6}$$

$$p_j \in \langle 0, 1 \rangle, \sum_{j=1}^{|S|} p_j = 1 \tag{7}$$

$$Gp^T \geq 0 \tag{8}$$

The (8) is a set of linear inequalities with G as a matrix of coefficients. G-matrix completely describes all possible actions of the players and their consequences. Algorithm to compute G-matrix will be described bellow. Z_j in (9) denotes one complex payoff of all players together in the j-th strategy profile.

$$Z_j = \sum_{i=1}^{N} w_i U_i(s_j) \tag{9}$$

There are generally three approaches to that: $w_i = 1$, $w_i = \frac{1}{N}$, w_i are different to each player (for example to normalize them if they are not similarly strong). These weights are for everyone to tune for his own particular application. There is absolutely no general recommendation for that.

Solving the LP-problem, we obtain an optimal point $(p_1, ..., p_{|S|})$, Z contains a (Pareto) optimal outcome for all players which is the highest possible and no player wishes to deviate. The vector $(p_1, p_2, ..., p_{|S|})$ is the wanted correlated equilibrium. Anyway, there is a strong influence of the linear constraints defining what *strategies*

will each player never play. The LP-solver constructs a problem domain defined by given inequalities modelling the basic Nash rule of equilibria (G-matrix) – the *i*-th player will not change his strategy j to k in s^* profile if he will not get better $U_i(k, s^*_{-i})$ than $U_i(j, s^*_{-i})$. And finally, the LP-solver will find the best profile with this constraint.
Computing the G-matrix

G-matrix collects all relative preferences of players in Γ game regarding their strategies and can be constructed by following rules:

(i) Rows of G-matrix are indexed ijk where i indexes a player, j his strategy and k his alternative strategy. The i-th player evaluates how his profit is going to change if he moves from j strategy to k strategy. There are $\sum_{i=1}^{N} |S_i|.(|S_i|-1)$ of rows in the G-matrix.

(ii) Columns of G are particular strategy profiles $s \in S$ of the Γ game. There are $\Pi_{i=1}^{N} |S_i|$ of columns.

(iii) Cell $g_{ijk,s}$ in the $G-$matrix at ijk row and column $s \in S$:

$$g_{ijk,s} = \begin{cases} U_i(s) - U_i((k, s_{-i})) & s_i = j \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

The G-matrix is a very simple structure (2D matrix) appropriate for further game-theoretic analysis of strategy space. The multi-dimensional problem is transformed into a 2D matrix. It simplifies the following analysis.

5.2 An example of solving a game

Let us have a two player game $\Gamma = (\{1,2\}; \{a,b\}, \{c,d\}; U_1, U_2)$ with payoffs written in Figure 3. The G-matrix for this game is computed in Figure 4 with profiles ordering (ac, ad, bc, bd). This system will generate a following LP problem:

$$MAXIMISE : Z = 12p_1 + 12p_2 + 9p_3 + 10p_4 \tag{11}$$

$$p_{1..4} \in \langle 0, 1 \rangle \tag{12}$$

$$\sum_{i=1}^{4} p_i = 1 \tag{13}$$

$$5p_1 + p_2 \geq 0 \tag{14}$$

$$-5p_3 - p_4 \geq 0 \tag{15}$$

$$-2p_1 + p_3 \geq 0 \tag{16}$$

$$2p_2 - p_4 \geq 0 \tag{17}$$

The Nash equilibrium is clearly ad. From (15) we see that $p_{3,4} = 0$, then from (16) that $p_1 = 0$ and so, (14) together with (13) gives $p_2 = 1$. We interpret the result $(0,1,0,0)$ that the profile ad wins with probability 100%.

Figure 3. Payoffs in the example

	c	*d*
a	10,2	8,4
b	5,4	7,3

Figure 4. G-Matrix in the example

	ac	*ad*	*bc*	*bd*
$a \to b$	5	1		
$b \to a$			-5	-1
$c \to d$	-2		1	
$d \to c$		2		-1

5.3 Iterative elimination of dominated strategies

We have implemented a linear programming solver of correlated equilibrium using GLPK library (GLPK homepage 2008, GNU Linear Programming Kit). Solving CE has *theoretically* always a solution as it is just another view to Nash equilibrium in mixed strategies (Nash 1951). Moreover, solving this problem has a polynomial time complexity (Papadimitriou 2005). However, for a very large game, the computation takes too much time or computer memory. The LP-solver can get lost in some numerical instability too. Thus, getting a proper solution is not absolutely guaranteed in practice. This trouble can be cured by reducing the game to its smaller equivalent, as we will see now.

Iterative elimination through the G-matrix

We are going to explain this approach using the previous example (Figure 3). Let us examine the Figure 4. The first row shows that all payoffs are positive. It means that the row-player has his payoffs always higher when playing *a*-strategy rather then *b*-strategy. The rows with zero elements are redundant as they have no effect to the LP-problem solving.

The second row is more interesting, because all differences are negative (we do not care about the zero elements). It indicates that the utility when playing *b*-strategy is always worse then when playing *a*-strategy, thus *a* strongly dominates the *b*-strategy. No matter, if *b* is dominated by all other strategies of row player, to satisfy the constraints (12) and (15), probabilities assigned to *bc* and *bd* profiles must be zero, so $p_3 = p_4 = 0$. The negative row and corresponding columns, i.e. the LP variables, p_3 and p_4 are removed then.

The process may continue until the G-matrix is minimized, it means without nega-

tive rows (see Figure 5). In some cases, this elimination may lead to a single-profile state. Otherwise, we normally apply the LP-solver to the reduced game specification, it means to the reduced set of profiles (probability variables) and corresponding strategy sets.

Figure 5. Elimination steps

	ab	*ad*
$a \to b$	5	1
$c \to d$	-2	
$d \to c$		2

$\Rightarrow$

	ad
$a \to b$	1
$d \to c$	2

5.4 *G-solve*: An efficient algorithm for solving the correlated equilibria

Having this elimination procedure we may turn the order of computing activities from the former sequence of (i) computing the state space, (ii) eliminating dominated strategies, and (iii) equilibria computing to a new optimized algorithm where *computing the state space* is being done on-the-fly as a sub-part of the elimination procedure.

By our experience, the *G-solve* terminates in shorter time than the classical approach (unfortunately, it has not been studied deeply at a theoretical level) and it is significantly less memory consuming.[2] Let us define following data structures and data types:

(i) $valid : S \to Boolean$ is a Boolean array displaying what profiles are valid or non-valid (zero/non-zero probability). Initially, all profiles are valid.

(ii) $umap : S \to \mathbb{R}^N$ is a dynamic dictionary (items can be added and removed during the computation) assigning payoff vectors to particular profiles. Initially, *umap* is empty.

(iii) $dominated : Q \times S_i \to Boolean$ is a boolean array displaying what strategies are already found to be dominated. Initially, all are set *False*.

(iv) type $profList$ is a list of tuples $(S, \mathbb{R})$.

(v) type $gRow = (pos : profList, neg : profList)$

(vi) $G[i, j, k]$ is a dynamic list of *gRow* indexed by (i, j, k). G represents the G-matrix.

The *G-solve* algorithm (Algorithm 3) is started and goes through all players of the Γ game and all variants of their behavior. The whole algorithm is split into parts, Algorithm 3, 5, 6 and 7 (see the Appendix), to make it easier to study. At the end, remaining G list and *valid* array contain the resulting minimized game Γ_r.

[2] The algorithm has been developed and tested on PC with 8xCPU Xeon 2.66GHz and 16 GB RAM.

Algorithm 3 G-solve – main part

```
for i in Q:
for j in S[i]:
for k in S[i]:
 if (j!=k):
  if (not(dominated[i][j] or dominated[i][k])):
     row = solveRow(i,j,k);
     if (row != ([],[])):
        if (row.pos != []):
          G[i,j,k] := row;
        else:
          dominated[i][k] := true
          for (s,r) in row.neg:
            disableProf(i,s);
```

Let $\Gamma_r = (Q; S_{r1}, ..., S_{rN}; U_{r1}, ..., U_{r2})$ is the minimized game Γ_r. Strategy sets are reduced to S_{ri} in (18) and only the utility functions (19) are selected to the new game.

$$S_{ri} = \{s_i | s \in S \wedge valid[s]\} \tag{18}$$

$$U_{ri}(s) = U_i(s); \forall s \in S_r \tag{19}$$

The G-solve elimination may terminate in state when $S_r = \{s^*\}$. In that case, s^* is the equilibrium (in strictly dominant strategies). Otherwise, if $|S_{ri}| > 1$ for some player $i \in Q$, the CE-solver can be applied to the resulting Γ_r game. The G-matrix for Γ_r is already completed in $G[i, j, k]$, just some mapping from Γ state space to Γ_r state space has to be specified.

5.5 Meaning of this method

G-solve is an exact method of efficient computing the correlated equilibria in multi-player games. The method combines the computing of the game optimum together with computing the utility functions. In this way, only the relevant cells of state space are analyzed. It is a type of a heuristics which does not effect the *cellModel* and reduces the number of cells searched during the game analysis.

When comparing the time and memory complexity of computing CE without G-solve and with G-solve approach, we can see that complexity without G-solve defines the worst case complexity for the approach with G-solve enabled. The final complexity of the equilibrium determination may be improved depending on particular game and formulation of the internal model *cellModel*. This is very application specific.

The G-aolve method is suitable for computing the CE in relatively small games where the size of state space does not exceed $10^8 - 10^{10}$ cells. The method can be used as a terminating operation after another mechanism of state space reduction (for example FDDS in the following section).

Table 1. Demonstration of computing speed-up with G-solve enabled

Experiment number	Number of profiles	Execution time with G-solve enabled	Execution time without G-solve
n1	11,520	0.305 s	1.084 s
n2	155,520	5.9 s	14.8 s
n3	5,054,400	4 min 51 s	16 min 12 s
n4	15,300,000	13 min 54 s	memory shortage

As a demonstration of the method in operation, we provide a small experiment (Table 1). The cases (games) were generated out of our MCE model (see section 4.1). From the equilibrium determination point of view, just the number of strategic profiles is relevant. The part solving the correlated equilibria was executed as an independent program where its execution time is measured. The experiment was done on PC with 4xCPU AMD Opteron 2.8GHz and 32 GB RAM. We show the execution times with G-Solve operational and without G-Solve – the program builds the whole G-matrix with no reductions (see section 5.1).

We can see that the processing time with G-solve enabled is always about 3-4 times shorter. The last experiment *n*4 without G-solve did not terminate correctly as the memory requirements exceeded the computer's capacity (32 GB RAM).

Unfortunately, we are not able to compare our implementation of the correlated equilibrium solver (with or without the G-Solver reduction mechanism) with another computer implementation of such a solver. To our best knowledge, there is no published paper on technical details of this problem or a computer tool solving that.

6. Fast Detection of Dominant Strategies (FDDS)

Analyzing the strategy dominance was the starting idea of this method again. We expect the input state space S to be entered extremely wide ($|S| \simeq 10^{30}$ is often in our models) and that S will get significantly reduced. We assume that existence of strict dominance for some players is probable as well. When doing the practical experiments in multi-dimensional decision modeling (see (5)), it is not rare that a certain subpart of the decision variables $D_{i,dom} \subseteq D_i$ of a player i demonstrates a strictly dominant behavior. In such a case, the player makes his decision de facto just in $D_i \setminus D_{i,dom}$. For example, some players are sure that they make better when selling their total production to the domestic buyer rather then exporting that (then the sub-decisions "sell maximum home" strictly dominates "sell a part home and export the rest").

Let us assume the distinguishability of the players' strategies. The player i then exactly knows in any profile $s \in S$ if he wants to change his strategy s_i (if $s_i \notin BR(s_{-i})$) or not. If $BR_i(s_{-i}) = \{s_i'\}$, then the player i exactly knows the strategy s_i' to move into. Otherwise he decides one of $BR_i(s_{-i})$.

The following algorithm (FDDS) allows various forms of outputs including the fast

detection of dominant strategies, pure Nash equilibria, or detection of certain cycles demonstrating the mixed behavior (sources of mixed Nash equilibria). So called Graph of Reachable Profiles (GRP) is constructed during FDDS operation. Solution to the games comes out from the GRP analysis.

It is very important to emphasize at the very beginning, that determining the game equilibria is not the primarily goal of FDDS. The goal is to make a very fast and representative preview at the important strategies of the players. FDDS is a simulation method and hence its quality strongly depends on efforts to experiment with the model. There is definitely no absolute guarantee that the FDDS algorithm reducing the input game Γ to Γ_r transforms the input to its strategically equivalent game Γ_r. Proving that is currently not possible neither analytically nor experimentally (it might be possible for small games as a case verification). We are pretty sure that the behavior in Γ_r is not significantly far from Γ. It is the best available solution to the large games at the moment, and rather satisfactory for our applications.

Definition 7. Graph of Reachable Profiles of a given game $\Gamma = (Q;S;U)$ is a structure $GRP = [V,E]$, where

(i) V is a (finite) set of nodes (s,Q_a,Q_r), where $s \in S$; $Q_a,Q_r \subseteq Q; Q_a \cap Q_r = \emptyset$. Q_r is a subset of players who agree with the profile s. Similarly, Q_r are those who does not agree with s. Only players i having $s_i \in BR_i(s_{-i})$ do agree with the profile s.

(ii) $E \subseteq V \times V \times Q$ is a set of edges. Edges are relevant just for analyzing the graph topology (cycles, trees). The edges are restricted just for non-agreeing players, thus $\forall(v_1,v_2,i) \in E, v_1 = (s,Q_a,Q_r) : i \in Q_r$. The edge expresses the i-th player deviation from the v_1 node to the profile corresponding to the node v_2.

In the following text, we demonstrate the algorithm of computing the GRP for a given game and we introduce its analysis.

6.1 Analysis of GRP

Let us have a graph of reachable profiles $GRP = [V,E]$ of a given game $\Gamma = (Q;S;U)$. The set of profiles in (20) is called the *set of reachable profiles*. A node $v \in V$ is called *to be solved* if $Q_a \cup Q_r = Q$. The node $v \in V$ is a pure Nash equilibrium if $Q_a = Q$ (i.e. $Q_r = \emptyset$).

$$S_{res} = \{s|(s,q_a,q_r) \in V\} \tag{20}$$

If the *GRP* is topologically a tree and all its nodes are solved, the related game Γ has a solution in form of pure Nash equilibria. If the number of PNEs is greater then one, we shall compute their MNE complements using some other mathematical methods.

Studying cycles (or clusters of cycles) in GRPs is more difficult and it exceeds the range of this paper. We believe that their analysis can lead towards fast computation of mixed equilibria. Generally, a cycle in graph is a closed path with no other repeated node than the starting (ending) one. In the meaning of strategic behavior, a player i in a

profile s^0 does not agree with the profile and chooses another profile $s^1 = (BR_i(s^0), s^0_{-i})$. The next player j then continues to s^2 and so on until the starting player i moves back into s^0 again.

In the current state, there is no proper experimental and theoretical conclusion about logic connection between cycles and equilibria. The reduced game Γ_r is thus defined as a game within the set of reachable profiles S_{res} in (20) of Γ. So, Γ_r is defined as follows:

$$\Gamma_r = (Q; S_r; U_r); S_{r,i} = \{s_i | s \in S_{res}\} \tag{21}$$

We compute the equilibria based on Γ_r using conventional methods (for example G-solve).

6.2 Algorithm of GRP construction

A potential dominant strategy s_i of a player i will become evident in any strategic profile s. Thus, we may start with any randomly chosen profiles $s^0 \in S$ in our analysis. We study, if the player i would deviate in s^0 profile. The set of strategies given in (22) is a set of potential deviations of the player i.

$$B_i(s^0) = \{b \in S_i | U_i(b, s^0_{-i}) \geq U_i(s^0)\} \tag{22}$$

Clearly, $BR_i(s^0) \subseteq B_i(s^0)$. The player i will react in the profile s^0 moving into some $b \in BR_i(s^0)$. If $|BR_i(s^0)| > 1$ holds, we have to analyze concurrently more similar options of the player (branching). As it was mentioned many time before, it is highly preferable if the strategies are well distinguishable and such a state does not appear.

Inserting a node to the GRP structure

This procedure (see Algorithm 4) adds to $GRP = [V, E]$ all best-response strategies B of the player i playing from the current node $v = (s, Q_a, Q_r)$.

Algorithm 4 Inserting new nodes (*B,v,GRP*)

for b in B:
    if $s_i = b$ then add i to Q_a,
    else:
        add i to Q_r
        if exists $v' = (s', Q'_a, Q'_r)$ in V that $(b, s_{-i}) = s'$:
                add i to Q'_a
        else:
                add a new $v' = ((b, s_{-i}), \{i\}, \emptyset)$ to V
        add a new edge (v, v', i) to E

Main algorithm of GRP construction

The top view on the algorithm is as follows:

(i) Initialize the $GRP = [V_0, \emptyset]$ with randomly generated nodes. If $S_{rand} \subset S$ is a set of random profiles, the initial set of nodes is $V_0 = \{(s, \emptyset, \emptyset) | s \in S_{rand}\}$.

(ii) Select randomly a node $v = (s, Q_a, Q_r)$ from V which is *not solved* yet. If there is no such a node left, terminate the algorithm.

(iii) Compute $B := BR_i(s)$ using the Algorithm 1. Let i is a player randomly chosen in v from $Q \setminus (Q_a \cup Q_r)$. We would like to remind that this operation requires to invoke the $cellModel(s,C)$ for all $s \in \{(s_i, s_{-i}) | s_i \in S_i\}$. It means, $|S_i|$ times.

(iv) Insert all B chosen by the player i coming from v to the *GRP* structure (Algorithm 4).

(v) Go to step 2.

The algorithm terminates if all nodes of its *GRP* being constructed are solved, or if the number of nodes is equal to the cardinality of the set of all strategic profiles (the state space). Termination of the algorithm is thus guarantied (the computability). If we study this case in point of view of the algorithmic (time) complexity, the algorithm does not always terminates practically because the size of S may be huge.

The practical experiments conclude that expressing some explicit termination condition may have its reasons. There are two possible conditions for enforcing the algorithm to halt:

(i) $|V|$ exceeds a given limit,

(ii) age of *GRP* exceeds a given limit. Age of GRP is a length of uninterrupted sequence of the main algorithm iterating with no new node added to V.

The FDDS algorithm can be also scalable by number of initial randomly generated nodes. The practical experiments demonstrate that even one initial node can cause a large spread over the game state space. The algorithm is ready for parallel processing, so that the steps (ii)-(v) are done in parallel.

6.3 Meaning of the FDDS algorithm

The algorithm is suitable for a very fast analysis of a game state space, and for detection of good strategies given in (1). The FDDS algorithm is scalable. The computation complexity can be regulated by number of initial nodes, number of maximal nodes in the graph and number of steps with the graph unchanged.

A reduced game Γ_r is the output from the FDDS algorithm. Probability of strategic equivalence (see section 2.2) between Γ_r and the original Γ grows with the effort of experimenting. Let us emphasize once more that the *quality of the output is significantly influenced by the strategy distinguishability*. If the state space is well distinguishable as defined in (4), the FDDS analysis converges quickly.

7. Conclusion

The paper presented the methodology and algorithms which we are currently using in the design and development of computer models of commodity markets. The models

are intended for analysis of the markets and for forecasting of their further evolution in horizons of 1–15 years. There is surely a plenty of similar modeling techniques related to this research area. We have focused mostly to a very narrow part of them, specifically to the methods of state space reduction. The paper does not analyze a particular model or a case study. It is rather a general description of architecture of these models. Anyone should be able to build his own model following this methodology.

The models are conceived as a tool for massive experimenting. An user-experimenter works with them in the simulation manner, i.e. specifies his queries and obtains a new knowledge of the modeled system. Allowing the user to enter the experimental domain (strategy sets) as wide as possible was the very required feature of these models. A limited state space S may cause that some reasonable variant of the behavior is neglected. The experiment thus fails or ends up with wrong results.

When implementing the reduction techniques, we have to keep in mind the question, if the reduced and original game are strategically equivalent, i.e. that all strategically important strategies of the original game are included in the reduced game as well. We presented two methods: it is clear that the strategic equivalence is not damaged in G-solve method (in the meaning of correlated equilibria). This correspondence may be corrupted by the FDDS method, or to be more precise, the probability of non-correspondence gets close to zero as we spent the time in experimenting. The quality of the whole process is highly influenced by the proper design of *cellModel*. Every small detail of the player's strategy and the current context must be included in the computing of the player's payoff. It makes the differentiation between the strategies easier.

We presented a very wide framework of all actions heading towards the implementation of an operating game theoretical computer model. Some of the details were probably not discussed properly. However, the framework itself is a mosaic where any its internal component may be substituted by another one. For example, the equilibrium solver (Figure 2) can be concretized with an implementation of Nash-equilibrium solver, or correlated equilibrium solver (G-solve), or Stackelberg-equilibrium, or any user-specific way of predicting the players' behavior. The paper comes from the Computer science background and its main contribution stays in making the sophisticated algorithms of large problems and their use in computer simulations.

Acknowledgment The author would like to thank to anonymous referees for their helpful comments. This work has been supported by the Grant Agency of Czech Republic project No. GP102/06/P309 *Modeling and Simulation of Intelligent Systems* and the Czech Ministry of Education under the Research Plan No. MSM0021630528 *"Security-Oriented Research in Information Technology"*. The work has been done in cooperation with EGÚ Brno Ltd.

References

Aumann, R. (1974). Subjectivity and correlation in randomized strategies. *Journal of Mathematical Economics*, 1, 67–96.

Erev, I., Roth, A.E., Slonim, R.L. and Barron, G. (2002). Predictive value and the usefulness of game theoretic models. *International Journal of Forecasting*, 18, 359–368.

Green, K. C. (2002). Forecasting decisions in conflict situations: a comparison of game theory, role-playing, and unaided judgement. *International Journal of Forecasting*, 18, 321–344.

Gountis, V. P. and Bakirtzis, A. G. (2004). Efficient Determination of Cournot Equilibria in Electricity Markets. *IEEE Transactions on Power Systems*, 19, 1837–1844.

Gambit homepage (2008). http://gambit.sourceforge.net/

GLPK homepage (2008). http://www.gnu.org/software/glpk/

Hrubý, M. and Toufar, J. (2006). Modelling the Electricity Markets using Mathematical Game Theory. Proceedings of the 15th IASTED International Conference on APPLIED SIMULATION AND MODELLING, ACTA Press, 352–357.

Hrubý, M. (2007). Modelling the Structured and Complex Decision Situations. Proceedings of the 6th EUROSIM Congress on Modelling and Simulation, Ljubljana, p. 10.

Hrubý, M. and Čambala, P. (2008). Efficient Computing of Correlated Equilibria in Multi-Player Games. Proceedings of 12th International Conference on Artificial Intelligence and Soft Computing, Spain, 185–191.

Krause, T., Andersson, G., Ernst, D., Beck, E. V., Cherkaoui, R. and Germond, A. (2004). Nash Equilibria and Reinforcement Learning for Active Decision Maker Modelling in Power Markets. Proceedings of the 6th IAEE European Conference: Modelling in Energy Economics and Policy, Zurich, p. 6.

Krishna, V. (2002). *Auction Theory*. London, Elsevier.

Kwang-Ho, L. and Baldrick, R. (2003). Solving Three-Player Game by the Matrix Approach With Application to an Electric Power Market. *IEEE Transactions on Power Systems*, 18, 1573–1580.

Latorre, M. and Granville, S. (2003). The Stackelberg equilibrium applied to AC power system–A noninterior point algorithm. *IEEE Transaction on Power Systems*, 18, 611–618.

Myerson, R. B. (2004). *Game Theory: Analysis of Conflict*. Harvard, Harvard University Press.

Nash, J. (1951). Non-Cooperative Games. *Annals of Mathematics*, 54, 286–295.

Nisan, N., Roughgarden, T., Tardos, E. and Vazirani, V. V. (eds.) (2007). *Algorithmic Game Theory*. Cambridge, Cambridge University Press.

Osborne, M. and Rubinstein, A. (1994). *A course in game theory*. Cambridge, London, MIT Press.

Papadimitriou, Ch. (2005). Computing correlated equilibria in multi-player games. Proceedings of the thirty-seventh annual ACM symposium on Theory of computing, 49–56.

Samuelson, L. (2004). Modeling Knowledge in Economic Analysis. *Journal of Economic Literature*, 42, 367–403.

Viguier, L., Vielle, M., Haurie, A. and Bernard, A. (2006). A two-level computable equilibrium model to assess the strategic allocation of emission allowances within the European union. *Computers & Operations Research*, 33, 369–385.

Appendix. G-solve code

Algorithm 5 G-solve – solveRow(i,j,k)

```
pos := []; neg := [];
for s in S: # run this in parallel
  if (s[i]==j and valid[s]):
    s2 = s; s2[i] := k;
    if (valid[s2]):
       uj := getU(s,i);
       uk := getU(s2,i);
       if (uj != Nil and uk != Nil):
          diff := uj - uk;
          if (diff != 0):
             if (diff>0):
                pos.append( (s, diff) );
             else:
                neg.append( (s, diff) );
return (pos, neg);
```

Algorithm 6 G-solve – getU(s,i)

```
if (umap[s]==Nil):
  # this is the main computation load
  umap[s] := cellModel(s, C);
return umap[s][i];
```

Algorithm 7 G-solve – disableProf(it,s)

```
valid[s] := False
umap[s] := Nil # free the umap item
for i in [1,...,it]:
for j in S[i]:
for k in S[i]:
   row := G[i,j,k]
   row.neg.removeKeyIfPresent(s);
   row.pos.removeKeyIfPresent(s);
   if (row.pos == []):
      for (s2,r) in row.neg:
          disableProf(it, s2)
      G[i,j,k] := Nil
```